DISK FILE MANAGER

FUNCTIONAL DESCRIPTION

Prepared for: ATARI, INC

Prepared by: Harry B. Stewart
Neoteric
15816 San Benito Way
Los Gatos, Calif. 95030
(408) 356-6477

DISK FILE MANAGER
FUNCTIONAL DESCRIPTION

1. DISK FILE MANAGER GENERAL DESCRIPTION

The Disk File Manager has the purpose of providing the using programs with mass storage access in a high level file-oriented manner. By using the File Manager, the programmer need not concern himself with the problems of:

Directory Maintenance
Space allocation
File integrity
Disk integrity
Recovery from system crashes or power downs.
Optimizing disk use
Error recovery
Record blocking and deblocking

A disk has a directory of file names, each name corresponding to a file which is a collection of data on the disk. Each file has a beginning and an end and is further divided (usually) into elements called records. The user reads and writes data records to/from the file without regard to their true physical locations on the disk. All blocking and deblocking of data is done by the File Manager in buffers that it provides.

Commands are available to read and write file data, to rename files, to remove files from the directory and to obtain information about files; these commands are described in detail in Section 3.

2. DISK FILE MANAGER FEATURES

This section describes the basic capabilities of the File Manager. The file access methods, record types, data types, and record access methods will be identified and discussed.

2.1 File Access Methods

Two file access methods are provided: sequential and random. Sequential access allows the user to read or write data starting at the beginning of the file and to proceed sequentially toward the end of the file; random access allows the user to select any record contained in the file and read it or write to it, records being identified numerically by their relative position within the file.

A third access method (that is not file oriented) is physical level I/O; this allows the user to read or write any sector on the disk by referring to its "physical" location. This access method will be used only by system programs.

2.2 Record Types

Two types of data records are recognized by the File Manager: fixed length and variable length.

For files with fixed length records the record length is established at the time the file is created and may not be changed as long as the file is in existence. Each record is allocated space in the file for its fixed size, even if the amount of user supplied data in the record is less than that size.

For files with variable length records the record length is newly established for each individual record that is written. Records are "packed" onto the disk, so best utilization of space is made.

Note that random access is supported only for files with fixed length records.

2.3 Data Types

not required if other handlers don't use EOL termination for text.

Two types of data within records are recognized by the File Manager: text and binary.

For records with text data, special provisions are made to assure that records are terminated by a logical end-of-line (EOL); the user may write and read text records without knowing their length if he chooses, because the EOL is recognized by all concerned parties as being a record terminator.

For records with binary data, all reads and writes must be accompanied by length specifiers.

2.4 Record Access Methods

Two record access methods are provided: record aligned and character aligned.

Record aligned accesses have two aspects, the reading and writing of data. Each and every write operation creates a record (by definition) and each read operation reads all or part of a new record starting at the beginning. To restate the above in a negative way, record aligned reads and writes cannot be used to deal with record portions that do not include the start of the record.

Character aligned accesses allow the user to treat the file as a character stream. Any number of characters may be read or written and the following operation will continue where the prior one left off. Character aligned accesses may not, however, deal with portions of more than one record at a time.

2.5 Legal and Illegal Combinations

At this point you may be wondering if all combinations of types and access are legal and/or meaningful. The diagram below shows the illegal combinations as X's and the READ only combinations as $\overline{W}$.

	SEQUENTIAL		RANDOM		
Fixed Length	$\overline{W}$				Binary
					Text
Variable Length			X	X	
	$\overline{\overline{W}}$		X	X	Binary
	Character Aligned	Record Aligned		Character Aligned	

2.6 I/O Queueing

The request queueing capabilities have not been decided; it is not clear to the author whether request queueing is required and/or meaningful.

3. DISK FILE MANAGER PROGRAM SERVICES

This section describes the file I/O services that are provided through program calls. The communication of parameters will be via File Control Blocks (FCB's), but since the formats for these partially device independent structures have not been specified yet, the FCB layout will not be identified.

3.1 OPEN File (Output)

The function of OPEN (output) is to find the file of the given name on the specified volume, if it exists, or to create a directory entry for the new file, if it doesn't exist.

The following parameters are supplied by the calling program:

Volume identifier (drive number)
File name/extension/attributes
Access method (sequential or random)
Record type (fixed or variable length)
Data type (text or binary)
Record length, if fixed length specified
Append option - positions to the End-of-File so WRITEs will append to the end of the current file
OPEN only if existing option - OPENs the file only if it already exists in the volume
OPEN only if not existing option - OPENs the file only if the file does not exist in the volume.

Delete on CLOSE option

The following item is returned by the File Manager:

OPEN status (see Section 4.5)

The following restrictions are imposed upon the caller by the File Manager:

1. A new file will not be created if the access method is specified as random.

2. Random access will not be allowed to an existing file unless the file parameters and the OPEN parameters both specify fixed length records and the specified record lengths are identical.

3. Random access will not be allowed to an existing file unless the file data type is the same as the OPEN data type (text or binary).

4. The following rules relate to multiple OPENs for a single file:

   A file that is already OPEN may not be OPENed for output.

   A file that is already OPENed for output may not be OPENed again, for input or output.

   A file may be OPENed any number of times for input, subject to the restrictions above.

5. When an already existing file is OPENed for output, and the append option is not selected, WRITE operations will write directly over the old file data; any automatic file backup mechanism must be done at a level above the File Manager (using RENAME, DELETE, etc.).

3.2 OPEN File (Input)

The function of OPEN (Input) is to find the file of the given name on the specified volume, if it exists.

The following parameters are supplied by the calling program:

Volume identifier (drive number)
File name/extension
Access method (sequential or random)

The following items are returned by the File Manager:

OPEN status (see Section 4.5)
Record Type (fixed or variable length)
Record length, if fixed length
Data Type (text or binary)

The following restriction is imposed upon the caller by the File Manager:

1. A file will not be OPENed for random access unless the file record type is fixed length.

3.3 OPEN File (Input/Output)

The function of OPEN (input/output) is to find the file of the given name on the specified volume, if it exists. This operation is restricted to random access and allows the caller to both READ from and WRITE to the file.

See Section 3.1 for further discussion.

3.4 WRITE File

The function of WRITE is to place user supplied data onto the specified volume while maintaining the defined disk and file structures. The user may supply data as fixed length records, variable length records or as sub-records as short as a single byte.

The following parameters are supplied by the calling program:

Data buffer address
Data buffer length
Relative record number, if random access
Write mode (record aligned, character aligned, block)

The following item is returned by the File Manager:

WRITE status (see Section 4.5)

The following restrictions are imposed upon the caller by the File Manager:

1. The size of a file OPENed for random access may not be augmented via the WRITE process, i.e., the caller may not access records outside of the existing file.

2. Records may not exceed 255 bytes in length.

3.5 READ File

The function of READ is to get data from the specified volume and file and to place that data into a user supplied buffer. The user may read data as fixed length records, variable length records or as sub-records as short as a single byte.

The following parameters are supplied by the calling program:

Data buffer address
Data buffer length
Relative record number, if random access
Read mode (record aligned, character aligned, block)

The following items are returned by the File Manager:

READ status (see Section 4.5)
File data is moved to the caller's data buffer
Indication of number of data bytes moved to buffer

3.6 CLOSE File (Output)

The function of CLOSE (output) is to: define the extent of the specified file, put all remaining file pertinent data to the volume, release all resources required to support the file; and, in general, to make the file a permanent and accessible member of the volume of files.

There are no parameters supplied by the call program.

The following item is supplied by the File Manager:

CLOSE status (see section 4.5)

There are no restrictions on CLOSE; any OPEN filed may be CLOSED.

3.7 CLOSE File (Input)

The function of CLOSE (input) is to release all resources required to support the reading of the file.

There are no parameters supplied by the calling program.

The following item is supplied by the File Manager:

CLOSE status (see Section 4.5)

There are no restrictions on CLOSE, an OPEN file may be CLOSED.

3.8 REWIND File

The function of REWIND is to re-establish access to the beginning of a file without having the caller perform a CLOSE/OPEN sequence.

There are no parameters supplied by the calling program.

Seek Record In File (RANDOM)

The following item is supplied by the File Manager:

REWIND status (see Section 4.5)

The following restriction is imposed upon the caller by the File Manager:

1. REWIND has no meaning for files OPENed for output.

3.9 STATUS REQUEST

The function of STATUS REQUEST is to tell the caller whether a file of the specified volume and name exists, and if so, to supply information contained in the volume directory which relates to that file.

The following parameters are supplied by the calling program:

Volume identifier (drive number)
File name/extension

The following items are returned by the File Manager:

STATUS REQUEST status (see Section 4.5)
File attributes (user specified/variable)
File descriptors (structural/fixed)
Record size, if fixed record length — max length for WRITEs if variable.
Date field
First data block address
First pointer block address
Volume write protect status
File size
OPEN/CLOSED status

A special provision will be made whereby the caller may also obtain volume status, which will include the following:

Volume name
Number of files in volume
Number of allocated blocks in volume
Number of free blocks in volume
Number of bad blocks in volume
Number of remapped blocks in volume (see section 5.6)

3.10 RENAME File

The function of RENAME is to alter the name and/or extension and/or attributes of a given file.

The following parameters are supplied by the calling program:

Volume identifier (drive number)
File name/extension (current)
File name/extension/attributes (desired)

The following item is returned by the File Manager:

RENAME status (see Section 4.5)

The following restriction is imposed upon the caller by the File Manager:

1. A file that is OPEN may not be RENAMEd.

3.11 DELETE File

The function of DELETE is to remove the specified file from the designated volume, deallocating all disk resources that were unique to that file and removing the name from the volume directory.

The following parameters are supplied by the calling file:

Volume identifier (drive number)
File name/extension

The following item is returned by the File Manager:

DELETE status (see Section 4.5)

The following restrictions are imposed upon the caller by the File Manager:

1. A file that is OPEN may not be DELETEd.

2. A file that is "protected" by an attribute may not be DELETED: a "protected" file must be RENAMEd to be DELETEd.

3.12 READ/WRITE Data Handling

This section will discuss the various data transformations that occur when WRITEing and READing file data. Section 5.1 may be referred to also for specific file record formats.

3.12.1 Data Buffer to File Record Transformations & Restrictions (WRITE)

The following rules apply to data being written to a disk file; the rules are qualified as necessary by type and access options.

1. Variable length text records will have multiple contiguous blanks encoded to save space.

2. For text file WRITEs, the amount of data coming from the data buffer is limited by the buffer length, by the current record being filled or by the occurrence of an EOL character - whichever occurs first.

3. For binary file WRITEs, the amount of data coming from the data buffer is limited by the buffer length provided or by the current record being filled - whichever comes first.

record length =?

4. Character aligned WRITEs of binary data are not allowed.

5. Fixed length file records which are not filled with data by the caller will be zero filled by the File Manager.

6. Truncated text records will have the last data character replaced with an EOL.

3.12.2 File Record to Data Buffer Transformations (READ)

The following rules apply to data being read from a disk file; the rules are qualified as necessary by type and access options.

1. For text file READs, the amount of data coming to the data buffer is limited by the buffer being filled, by the current record being entirely read or by an EOL being read - whichever occurs first.

2. For binary file READs, the amount of data coming to the data buffer is limited by the buffer being filled or by the current record being entirely read - whichever occurs first.

3. When the user's data buffer is not entirely filled with new data, the remainder of the buffer is zero filled by the File Manager.

4. On READs of fixed length text records, the zero-fill in the record cannot be read to the data buffer; after the EOL is detected, the next READ will start with the next record (for character aligned or record aligned READs).

5. READs to variable length text records will de-compress the encoded multiple contiguous blanks.

6. Record aligned READs of text records will always be EOL terminated; character aligned READs of text records will not be EOL terminated unless the READ reached the logical end of record.

4. ERROR HANDLING

This section describes the general error handling capabilities of the File Manager and discusses some of the error recovery techniques that may be employed.

4.1 Program Call Parameter Checking

All program supplied parameters will be validity checked before being used by the File Manager, thus assuring that errors do not propogate nor compromise the integrity of a disk structure.

4.2 Control Structure Integrity Checking

All RAM based, long term control structures such as directories and bit maps will be sum-checked before each use to insure against "wild stores" and random bit drop-outs. Range checking of critical numbers will be in order also.

Control structures on the disk will also include checksums for an extra measure of safety.

4.3 Disk Read Error Recovery

Wherever possible the user should have the option of aborting or proceeding on the occurence of a disk read error. For data record read errors this seems fairly safe; but for reads involving structural elements, great care must be taken if proceed is allowed.

4.4 Disk Write Error Recovery

Disk write errors will be handled by a remapping capability; a certain number of sectors will be able to be logically re-assigned to other sectors on the volume. Once the provided number of remaps is exceeded, absolutely no recovery is possible from any new disk write error and the file involved will be CLOSEd by the File Manager.

Write errors on writes to any fixed address volume structure (such as the remap table itself) will be disastrous, as will writes to a disk structural element once the remap table is full.

Once a sector is found to be unwriteable, it will be flagged as being unusable, and no attempt will be made to use that sector again until the track containing that sector is successfully reformatted.

4.5 Itemized Error List

The table on the following page shows the specific error stati that have been identified so far and the operations to which each may apply.

5. VOLUME STRUCTURE

This section will discuss the volume structures that are present to support the features provided by the File Manager.

It is assumed that there is a single volume per disk and that a volume consists of only one disk. It is further assumed that a volume contains exactly one directory of files, and that all elements within the volume be contained entirely within the volume.

The structural elements that will be discussed in this section are:

Record
File
Directory of Files
Block Allocation Map
Bad Black Map
Remap Table
System Modules
Disk Descriptor

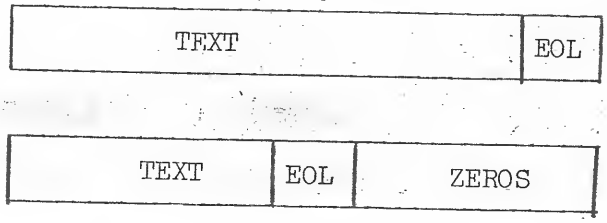
only binary?

5.1 Records

As discussed in prior sections there are fixed length and variable length records containing either text or binary data. Record size is independent of the sector size, and will be 255 bytes or less. Records will be stored in sectors with no dead space (records are not sector aligned). Several examples of record storage formats are shown below.

5.1.1. Fixed Length Text Records

All text records are terminated with an End-of-Line (EOL) Character*; if not by the WRITEr, then by the File Manager. WRITE records longer than the file record size will be truncated and terminated; WRITE records shorter than the file record size will be terminated and the remaining portion of the file record will be zero filled; and a WRITE record of exactly the file record size will be terminated. Two cases of file records result as shown below.



*Probably a carriage return

	OPEN (OUTPUT	OPEN (INPUT)	WRITE	READ	CLOSE (OUTPUT)	CLOSE (INPUT)	REWIND	STATUS REQUEST	RENAME	DELETE
Directory Read/Format Error	+	+					+	+	+	+
Directory Write Error	+				+				+	+
F.D. Drive Not Ready	+	+	+	+	+	+	+	+	+	+
Non-existent Drive #	+	+						+	+	+
File name/extension not found		+						+	+	+
Invalid File name/extension	+	+						+	+	+
New File Created	+									
Disk Write Protected	+		+						+	+
File Write Protected	+									+
File Open	+	+							+	+
File Not Open			+	+	+	+	+			
Disk Read Error ($\overline{\text{Directory}}$)	+		+	+				+		+
Disk Write Error ($\overline{\text{Directory}}$)	+		+		+					+
End-of-File				+						
End-of Volume (no free sectors)	+		+							
Invalid Record # (Random)			+	+						
Duplicate File name/ext. in volume	+								+	
Operation Invalid for Open Read	+	+	+		X			+	+	+
Operation Invalid for Open Write	+	+		+		X	+	+	+	+
Truncated Record			+	+						
End-of-Record				+						
Inadequate Memory	+	+								

5.1.2 Fixed Length Binary Records

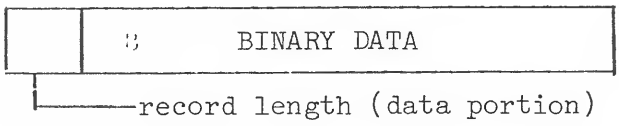
Binary records are unterminated. WRITE records longer than the file record size will be truncated and WRITE records shorter than the file record size will result in file record zero fill of the remaining portion. Two cases of file records result as shown below:

5.1.3 Variable Length Text Records

A variable length text record consists of an arbitarily long string of text characters terminated by an EOL.

5.1.4 Variable Length Binary Records

A variable length binary record consists of a record length followed by the binary data.



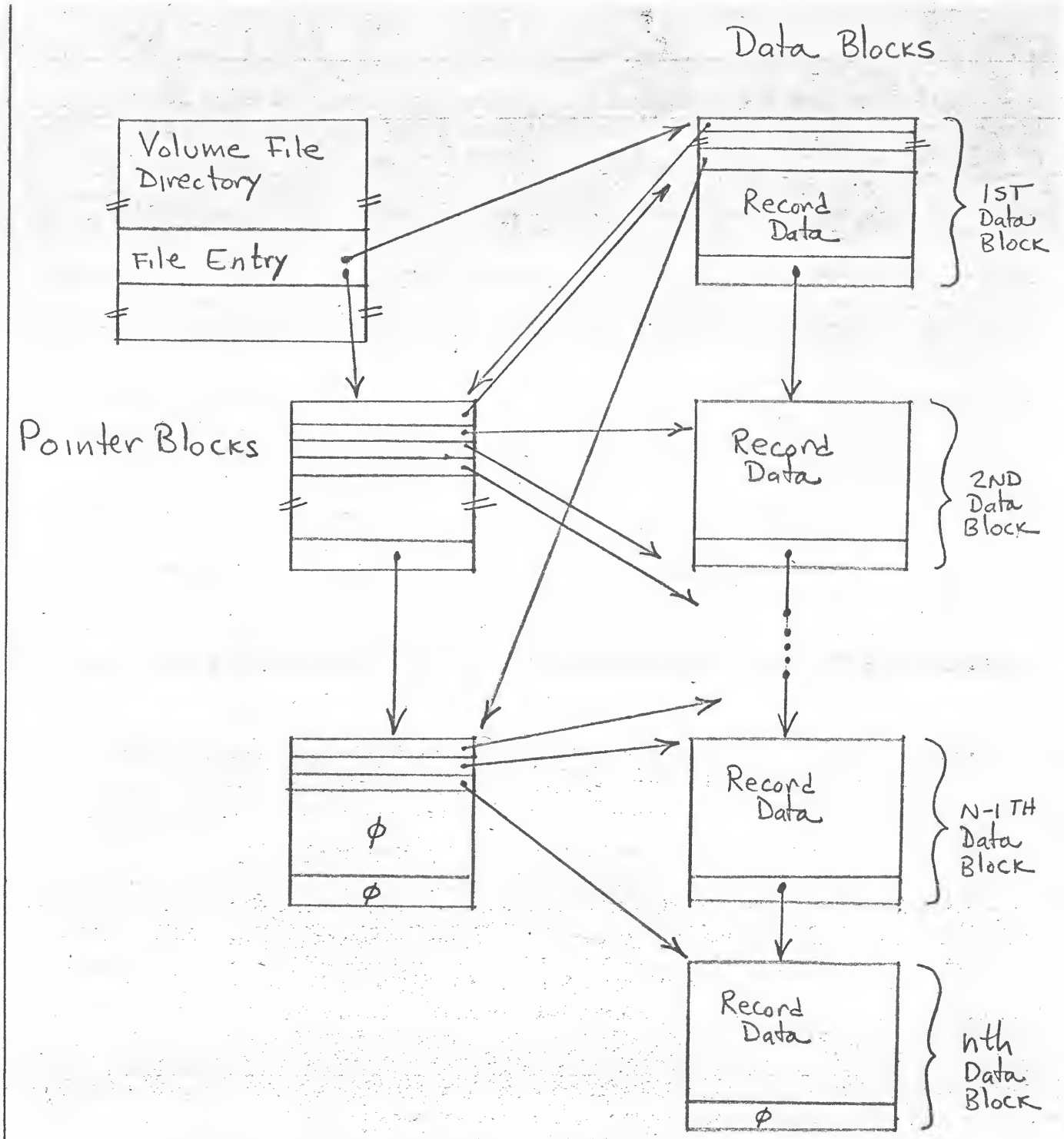
5.2 File

A file is a collection of zero or more records plus the support structures that are required to access the records. The diagram on the next page shows the structural elements of a typical file. Note, that block and sector are interchangeable terms as are disk address and pointer

The directory entry for a file contains the disk address of the first data block for the file. That data block in turn points to 2nd data block, and so on to the last data block of the file.

pointer blocks optional

TYPICAL FILE STRUCTURE



The directory entry for a file also contains the disk address of the first pointer block for the file; pointer blocks are chained in an identical manner to data blocks. Pointer blocks, instead of containing record data, contain the disk addresses of every data block in the file. Pointer blocks have at least 3 uses in the system.

1. They provide a means for fast random access without requiring contiguous file data block allocation on disk.

2. They provide a redundant data block ordering which can be used for file integrity checking (and correcting).

3. They provide an alternate way of supporting sequential access in the case of a data block read error, where the next block pointer may not be relied upon.

The first data block contains the disk addresses of all of the pointer blocks. This list of 12 double byte pointers is provided to add desired redundancy to the structure and as an essential aid to random access using large files (where an entire set of pointer blocks could not practically be contained in RAM, and sequential access of the pointer blocks would be extremely time consuming).

In addition, data and pointer blocks may contain additional overhead such as checksums, and file identifiers, but these design decisions will be deferred for now. Note also that a block is synonymous with a sector only because it appears convenient; a block could easily be 2 or more sectors.

5.3 Volume Directory of Files

The volume directory will contain information specific to that volume as well as the information relating to each file in the volume. The volume specific data will include:

Volume name (up to 32 bytes of text)
Volume attributes - these correspond to the file attributes, but override them when set.

The file specific data will include:

File name/extension - up to 16 bytes of file name/extension
File attributes - one byte of encoded attributes:
    delete inhibit
    write protect
    invisible for directory listing
File descriptors - one byte of encoded descriptors:
    text/binary records
    fixed/variable records
    open/closed file

End-of-File position (byte within last block) - one byte
First data block address - 2 bytes
First pointer block address - 2 bytes
Record size (fixed length only) - one byte
Last WRITE date - 2 bytes of encoded day/month/year

The directory resides in a binary file which is itself included in the directory, thus the size of the directory is not fixed. Access to this file will be limited to system programs.

A backup directory will be provided which will provide a fall back point in the case of the loss of the main directory. The backup directory will be updated automatically at power-up time or in response to an operator "backup" command.

5.4 Block Allocation Map

The allocation and deallocation of blocks within each volume will be maintained using a bit map, with one bit assigned per block. With the current disk size of 40 tracks of 18 sectors each, and a block equalling a sector, 90 bytes will completely map the disk.

The block allocation map may or may not be a file, pending further design.

5.5 Bad Block Map

The cataloguing of bad blocks within the volume is accomplished with a bit map which is identical in format to the block allocation map. When a block is found to be bad, its map bit is set, thus assuring that it will never be re-allocated in the future (unless a reformat of that block corrects the problem and resets the corresponding map bit).

The bad block map may or may not be a file, pending further design.

5.6 Remap Table

The disk address remap table is used to provide a certain amount of automatic error recovery without the bother of modifying block pointers in already existing files. This table will contain the disk address of every block which has been remapped and the disk address of the new location of the block. When a write error occurs, the failing block is flagged as bad in the bad block map and an alternate block is allocated; then the remap table is updated to show the new logical mapping for that bad block.

The remap table must be referenced by the File Manager for all file I/O operations, if it is not empty. It is feasible to provide a utility to empty the remap table by updating all block pointers within effected structures.

5.7 System Modules

One or more system modules will reside on each volume which will contain either a specific file bootstrap or a loadable version of the resident portion of the disk operating system.

System modules will either be in fixed locations on the disk or will be accessable through the disk descriptor. In either case, the system modules will be standard files.

5.8 Disk Descriptor

The disk descriptor will provide physical information about the disk; the following items may be included:

Size of disk (number of sectors)
Size of sectors (number of bytes)
Size of block (number of sectors)
Location of -
    Directory
    Maps
    System Module(s)

6. IMPLEMENTATION NOTES

This section provides additional information for the File Manager's implementor(s); the items discussed herein are not required for an understanding of how to use the File Manager, but are certainly required if one is to accomplish the design of the File Manager.

6.1 Minimization of Disk Accesses

Because floppy disks are inherently slow, an attempt must be made to minimize disk accesses. To this end, some assumptions have been made regarding the update of volume and file structural elements.

1. The volume directory will be written to disk whenever a file is OPENed for output and whenever a file OPENed for output is CLOSEd.

2. The block allocation map will be written to disk only at the time of a file CLOSE, and only if the map was modified since the last update.

3. File pointer blocks will be written to disk only when they become full or the file is CLOSEd.

4. The pointer block pointers in the first data block will be updated only when the file is CLOSEd.

5. When a block fills and another block must be allocated, the new block will be allocated before the old block is written so that the next-block-pointer may be present at the time the old block is written. This technique will be referred to as "pre-allocation".

Note that the cumulative effect of all of these optimizations is to make file reconstruction a bit of a task in the event of power failure with open files. However, since the data block pointers are all present, the pointer blocks may be found and reconstructed, the block allocation map reconstructed, the pointer block pointers derived and the file closed - assuming one can detect the end of the file. In fact, one can detect the end of file, because all unused blocks are to be zero filled - initially when the disk is first formatted and later when files are DELETED and the file blocks deallocated. It is anticipated that a disk controller command to zero a sector will be provided, which will preclude the necessity of sending a sector's worth of zeros down the serial bus.

6.2 Unique Volume Identification

To prevent the user from compromising the data integrity of disks by switching disks while files are open, the File Manager must have a way of recognizing one volume from another.

Some of the possible ways of identifying swapped disks are:

1. Multi-byte pseduo-random number for each disk generated at format time and never altered. Read when a file is opened and verified before file is closed.

2. Match the "in memory" file information with the volume directory information before the file is closed; for example: is the file name still in the directory at the same location as when the file was OPENed, does the descriptive information still match, is the volume name the same.

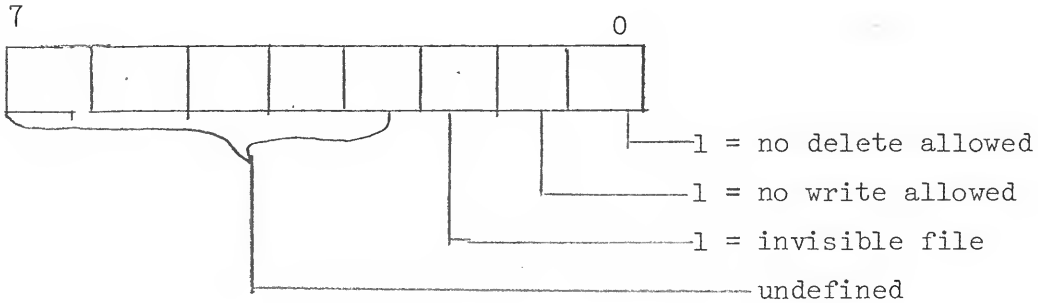
3. Provide a "volume open/close" flag or a counter of open files that is written to disk whenever a change of state occurs.

6.3 Program Parameter Formats

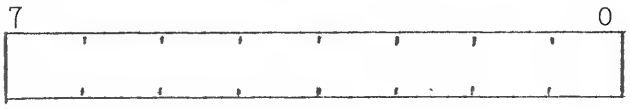
Although the File Control Block (format) has not been specified it is possible to describe the formats of the various parameters that will be used therein. In the paragraphs to follow, the parameter descriptions refer to their usage in a program call and do not pertain necessarily to the disk storage format of the same parameters.

1. filename/extension - the filename/extension is expressed as a string of up to 17 ASCII characters, terminated by an EOL character. The extension is delimited from the filename by the "/" character, when an extension is specified.

2. attribute - the file attribute is specified as a single byte which is bit encoded as shown below:

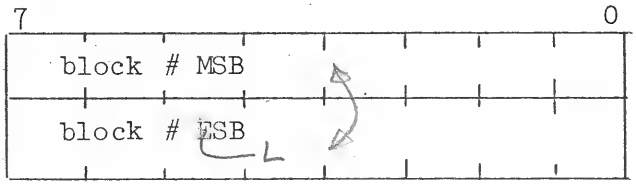


3. drive number - the volume drive number is specified as a single byte which may legally contain the values 1-9, and is further restricted by the actual number of drives installed in a given system.



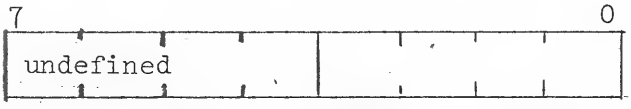
drive number

4. block number - the volume block number is specified as a double byte which may legally contain the values 1-720 (in binary); the byte with the lower memory address contains the most-significant-bits of the number.



not same as CP/M address storage format!!

5. command - the current command to be performed is encoded in 4 bits; the actual encoding is deferred to a later document.

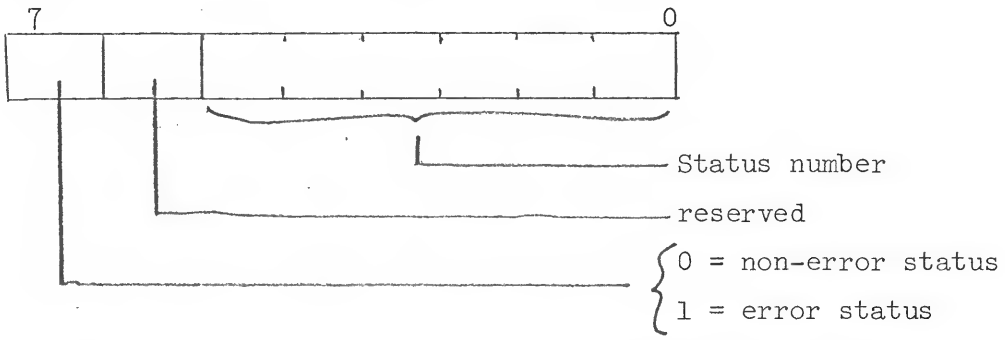


command code

6. Options - the command options are encoded to 4 bits which are command specific in format; the command and option fields will probably reside in a common byte.

Options

7. Status - the operation status is returned to the caller by the File Manager and is formatted as shown below:



If the entire byte is zero, this indicates that the requested operation has not completed yet.

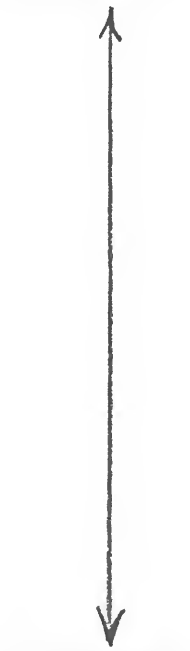
8. record length - the record length, when specified, is a single byte which may legally contain the values 1-255.

6.4. Filename/Extension Wildcarding

Processing of filename/extension wildcarding will be done by a utility that is not part of the File Manager; the File Manager places no special significance on the asterisk character.

APPENDIX A

CHAINED VS. CONTIGUOUS FILE ALLOCATION



CONTIGUOUS	CHAINED
Simple to design & implement	User's view is very simple & friendly
	Files are extendable
	Utilizes all of disk efficiently
Faster random access I/O	
	Not as sensitive to bad sectors
Sensitive to bad sectors	Slower access time
Inneffecient utilization of disk space due to "checkerboarding" of disk files	
Inefficient utilization of disk space due to unused space in each file	
Files not extendable	More complex design and implementation